TU Clausthal

Clausthal University of Technology

A Road Map of Updating in ASP

J. C. Acosta Guadarrama

IfI Technical Report Series IfI-07-16

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Wojciech Jamroga
Contact: wjamroga@in.tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

A Road Map of Updating in ASP

J. C. Acosta Guadarrama*

Institute of Computer Science,
TU-Clausthal, Germany
guadarrama@in.tu-clausthal.de

Abstract

As one of the major and traditional topics of Artificial Intelligence over many years, knowledge representation and reasoning has proved to be a strong theoretical framework for Logic Programming to manage dynamic knowledge bases. In this report, we go through current and some of those past proposals to update ASP programs, by analysing their features and identifying challenges to represent correct evolving knowledge.

1 Introduction

As one of the major and traditional topics in Artificial Intelligence over the last years, *knowledge representation and reasoning* has proved to be a strong theoretical framework to manage knowledge bases. As a result, this particular topic has become more widely applied in administration of knowledge bases of intelligent (rational) agents, especially in situations of incomplete knowledge from a changing environment, and this area of research is known in the literature as *belief updates*.

The history of *semantics for updates* of logic programs is rather long. Indeed, it starts in the days of some of the first versions of `Prolog` with its commands `assert` and `retract`. However, sooner they started to get inconsistencies and other (unexpected) *side effects*. It was also time of research on *databases* with publications like [FUV83], and in particular for *logical databases*: [Win90, FKUV86, Fag95]. Nevertheless, some of the first formalisms to carry out proper changes to monotonic theories have been originally studied by [AGM85, KM89, Mak88, KM91, Leh92], while in the non-monotonic side by [KLM90, Mak94, LM92].

Some years later, [GL88] formulated the *Stable Models Semantics* (also refereed as *Answer Sets Semantics*, SM or simply *ASP*), and more concrete proposals arose within that framework, aimed at the problem of updating knowledge: [ZF95, Zha01, Zha06, SI99, SI03, ALP$^+$99, EFST01, EFST02, EFST00b, EFST00a, OZ03].

*This project is mainly supported by a CONACYT Doctorate Grant.

In this technical report, we go through current and some of those past proposals to update logic programs (or alike), by pointing out features as well as some of their limitations to represent correct evolving knowledge.

2 Eiter et alia

To the best of my knowledge, [EFST02] achieve the most complete survey of all known semantics for updating logic programs, by gathering relevant postulates and principles from the literature. This approach first appeared in [EFST00a] with a vast study of well-known and well-accepted postulates and properties, and later refined in [EFST02] and extended to be a main component in more general problems like agents in [EFST05] or preferences in [EFLP02], and they also implemented a solver available at `www.kr.tuwien.ac.at/staff/giuliana/project.html#Download` that is the main engine of an experimental graphical front end from us at `www2.in.tu-clausthal.de/~guadarrama/updates/upd.html`.

One of the main assets of [EFST02]'s proposal, as already mentioned, is being one of the first[1] or even the first to realise a deep study of the literature of belief change of logic programs, in particular in ASP.

[EFST02] formulate a natural definition for updating logic program sequences on a restricted Answer Sets language by rejecting rules under a *causal-rejection principle*. The principle is due to [ALP+99] that later, however, turned out to be *counterintuitive*, even to themselves: see [EFST05, ABBL05, OZ03]. This "counter-intuition" comes from their strong dependency in the syntax of programs, according to [EFST05] Section 3 presents further discussion about this claim.

In particular, the natural formula under which [EFST05] analyse and describe update properties comes from [EFST02] as follows.

Given an update sequence $\mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n)$, with $n \leq 2$, over a set of atoms $\mathcal{A}$, assume $\mathcal{A}^*$ as an extension of $\mathcal{A}$ by new pair-wise unique atoms $\mathsf{rej}(\rho)$; α_i, for each rule ρ occurring in Π; each atom $\alpha \in \mathcal{A}$, and $1 \leq i \leq n$. An injective naming function $\mathsf{Name}(\cdot, \cdot)$ is also assumed, which assigns to each rule ρ in a program Π_i a unique name, $\mathsf{Name}(\rho, \Pi_i)$, provided that $\mathsf{Name}(\rho, \Pi_i) \neq \mathsf{Name}(\rho', \Pi_j)$ whenever $i \neq j$. Finally, for a literal ℓ, ℓ_i denotes the result of replacing an atomic formula α of ℓ by α_i.

The intuitive idea of $\mathsf{rej}(\rho)$ is that of an atom that blocks (rejects or *inhibits*) a related rule ρ when true, provided that there is another more recent rule ρ' with *conflicting information*.

Definition 1 (Update program [EFST02])**.** *Given an update sequence*

$$\mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n)$$

over a set of atoms $\mathcal{A}$, the update program $\mathbf{\Pi}_\lhd = (\Pi_1 \lhd \cdots \lhd \Pi_n)$ *over $\mathcal{A}^*$ consists of the following items:*

[1] They are the first, to the best of my knowledge.

(i) all constraints in Π_i, $1 \leq i \leq n$;

(ii) for each $\rho \in \Pi_i$, $1 \leq i \leq n$:

$$\ell_i \quad \leftarrow \quad \mathit{Body}(\rho), \text{not } \mathit{rej}(\rho) \qquad \text{if } \mathit{Head}(\rho) = \ell;$$

(iii) for each $\rho \in \Pi_i$, $1 \leq i < n$:

$$\mathit{rej}(\rho) \quad \leftarrow \quad \mathit{Body}(\rho), \neg\ell_{i+1} \qquad \text{if } \mathit{Head}(\rho) = \ell;$$

(iv) for each literal ℓ *occurring in* $\mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ $(1 \leq i \leq n)$:

$$\ell_i \leftarrow \ell_{i+1}; \qquad \ell \leftarrow \ell_i.$$

Note that in *(iv)* the authors write ℓ_1 rather than ℓ_i. Moreover, at the same *(iv)* they write $1 \leq i < n$ instead of $1 \leq i \leq n$. [Zha06] also detected these errors. Lastly, they do not state how to treat double negations that might happen in *(iii)*.

Next, [EFST02] define the intended answer sets of an update sequence $\mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ in terms of the answer sets of $\mathbf{\Pi}_\lhd = (\Pi_1 \lhd \cdots \lhd \Pi_n)$. In other words, the models are back to the original alphabet by filter them out with the original atoms:

Definition 2 (Answer sets of an update sequence [EFST02])**.** *Let*

$$\mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n)$$

be an update sequence over a set of atoms $\mathcal{A}$*. Then,* $\mathcal{S} \subseteq \mathit{Lit}_\mathcal{A}$ *is an* update answer set *of* $\mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ *if and only if* $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ *for some answer set* $\mathcal{S}'$ *of* $\mathbf{\Pi}_\lhd = (\Pi_1 \lhd \cdots \lhd \Pi_n)$*. The collection of all update answer sets of* $\mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ *is denoted by* $\mathcal{U}(\mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n))$.

There is a solver available for downloading at `www.kr.tuwien.ac.at/staff/giuliana/project.html#Download` and I have installed it to run online at `www2.in.tu-clausthal.de/~guadarrama/updates/upd.html`, which also provides a graphic-oriented interface on the server itself. Naturally, no download or installation is necessary to run it.

Supposing the corrected semantics is what the authors wanted, computing their following example is possible:

Example 1 ([EFST02])**.** *Assume a daily update regarding an energy flaw represented by the sequence* (Π_1, Π_2) *where*

$$\begin{aligned}
\Pi_1 = \{ \mathit{sleep} &\leftarrow \mathit{night}, \text{not } \mathit{tvon} \\
\mathit{night} &\leftarrow \top \\
\mathit{watchtv} &\leftarrow \mathit{tvon} \\
\mathit{tvon} &\leftarrow \top \} \\
\Pi_2 = \{ \neg\mathit{tvon} &\leftarrow \mathit{powerfailure} \\
\mathit{powerfailure} &\leftarrow \top \}
\end{aligned}$$

by Definition 1, the update program $\mathbf{\Pi}_{\lhd} = (\Pi_1 \lhd \cdots \lhd \Pi_n)$ *consists of rules (1)–(16):*

$$sleep_1 \leftarrow night, \text{not } tvon, \text{not } \mathit{rej}(\rho_1) \quad (1)$$
$$night_1 \leftarrow \text{not } \mathit{rej}(\rho_2) \quad (2)$$
$$watchtv_1 \leftarrow tvon, \text{not } \mathit{rej}(\rho_3) \quad (3)$$
$$tvon_1 \leftarrow \text{not } \mathit{rej}(\rho_4) \quad (4)$$
$$\neg tvon_2 \leftarrow powerfailure, \text{not } \mathit{rej}(\rho_5) \quad (5)$$
$$powerfailure_2 \leftarrow \text{not } \mathit{rej}(\rho_6) \quad (6)$$

$$\mathit{rej}(\rho_1) \leftarrow night, \text{not } tvon, \neg sleep_2 \quad (7)$$
$$\mathit{rej}(\rho_2) \leftarrow \neg night_2 \quad (8)$$
$$\mathit{rej}(\rho_3) \leftarrow tvon, \neg watchtv_2 \quad (9)$$
$$\mathit{rej}(\rho_4) \leftarrow \neg tvon_2 \quad (10)$$

$$sleep_1 \leftarrow sleep_2 \qquad sleep \leftarrow sleep_1 \quad (11)$$
$$night_1 \leftarrow night_2 \qquad night \leftarrow night_1 \quad (12)$$
$$tvon_1 \leftarrow tvon_2 \qquad tvon \leftarrow tvon_1 \quad (13)$$
$$watchtv_1 \leftarrow watchtv_2 \qquad watchtv \leftarrow watchtv_1 \quad (14)$$
$$\neg tvon_2 \leftarrow \neg tvon_3 \qquad \neg tvon \leftarrow \neg tvon_2 \quad (15)$$
$$powerfailure_2 \leftarrow powerfailure_3 \qquad powerfailure \leftarrow powerfailure_2 \quad (16)$$

whose unique answer set is

$$\{sleep_1, night, night_1, \mathit{rej}(\rho_4), \neg tvon_2, powerfailure, powerfailure_2, sleep, \neg tvon\}$$

and its ***update answer*** *set is easily obtained:* $\{night, powerfailure, sleep, \neg tvon\}$.

However, by following the original Definition 2 in [EFST02], rules (15)–(16) would not exist, for i should also equal n in Definition 1 item *(iv)*, and the answer set of the resulting program is

$$\{night, tvon, night_1, watchtv_1, tvon_1, powerfailure_2, watchtv\} \quad (17)$$

that means the *TV is on* and the agent is watching it. On the other hand, by changing i to be within the range I suggest and by leaving the second rule in *(iv)* as the original definition, that is to say, $\ell \leftarrow \ell_1$, the resulting program would have rules

$$\neg tvon_2 \leftarrow \neg tvon_3 \qquad \neg tvon \leftarrow \neg tvon_1$$
$$powerfailure_2 \leftarrow powerfailure_3 \qquad powerfailure \leftarrow powerfailure_1$$

instead of rules (15)–(16) and would have the same strange answer set in (17).

In other words, by following the original restriction presented in *(iv)* in Definition 2 in [EFST02], $powerfailure \notin \mathbf{\Pi} = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ when $1 \leq i < n$. Moreover, if the second rule in *(iv)* of Definition 1 was $\ell \leftarrow \ell_1$, a strange answer set would result:

$$\{night, night_1, powerfailure_2, watchtv_1, watchtv, tvon_1, tvon\}$$

Anyway, their solver at `www.kr.tuwien.ac.at/staff/giuliana/project.html#Download` *seems*[2] to behave well. Unfortunately, their solver does not show intermediate transformations to figure out the correct semantic parameters so that I can give a precise statement. Notice that I have only provided a frond end to execute their solver in the web with a graphical interface, and I employed the latter as the main engine of my front end: `www2.in.tu-clausthal.de/~guadarrama/updates/upd.html`.

Back to the corrections I suppose, let us complete Example 1:

Example 2 (Continued from Example 1)**.** *Make a second update to the sequence in Example 1 with the program* $\Pi_3 = \{\neg powerfailure\}$*. Accordingly, the new answer set of the resulting update program is*

$$\{tvon_1, tvon, night_1, night, watchtv_1, watchtv, \mathsf{rej}(\rho_6), \neg powerfailure_3, \neg powerfailure\}$$

Finally, by Definition 2, the corresponding **update answer sets** are

$$\mathcal{U}(\Pi_1, \Pi_2) = \{night, powerfailure, sleep, \neg tvon\}$$

and

$$\mathcal{U}(\Pi_1, \Pi_2, \Pi_3) = \{tvon, night, watchtv, \neg powerfailure\}$$

Despite the complete deep nice analysis [EFST02] make of known postulates and principles in the literature, one of the major shortcomings of their approach has to do with syntactic and semantic contents.

Take again, for instance, the example suggested by [ABBL05], that may produce *counterintuitive models*:

Example 3. *Suppose an agent who believes that when it is day there is no night and vice versa, and that there are stars when it is night and when there are no clouds. Finally, that at the current moment it is a fact that there are no stars. This short story may be coded into* Π_1 *as follows:*

$$\begin{aligned} \Pi_1 = \{ day &\leftarrow \text{not } night \\ night &\leftarrow \text{not } day \\ stars &\leftarrow night, \text{not } cloudy \\ \neg stars\} & \end{aligned}$$

[2] Unfortunately the sources are not available so as to confirm the latest definition.

whose unique answer set is $\{day, \neg stars\}$. *Later, the agent acquires new information stating that stars and constellations are the same thing, as coded in* Π_2. *So, if the agent updated* Π_1 *with program*

$$\begin{array}{rcl} \Pi_2 = \{stars & \leftarrow & constellations \\ constellations & \leftarrow & stars\} \end{array}$$

the expanded alphabet *of the two programs contains only one new* extra atom *with respect to* Π_1*:* $constellations$. *As the model of* Π_2 *is obviously the empty answer set,* $constellations$ *is considered* synonymous *of* $stars$ *by means of* Π_2, *and thus the update should not change the original knowledge base. However, the update introduces an* extra answer set *in many of the existing update semantics based on the* causal-rejection principle *see for example [ALP*$^+$*99, EFST02, ABBL05]:* $\{stars, constellations, night\}$ *which does not coincide with* common intuition.

To recapitulate, [EFST02] were *very* good in gathering postulates and principles from the literature and in analysing them in terms of their proposal. Their proposal, however, suffers from drawbacks owing to its reliance on the *causal-rejection principle* see [ALP$^+$99].

3 DyLP and Other Dialects

One of the earliest approaches in updating logic programs appeared in late 90's in [ALP$^+$99, ALP$^+$98] that was extended in an interesting language called LUPS by [APPP02], to specify *explicit updates* in programs on a semantics that they called *Dynamic Logic Programming* or DyLP —[ALP$^+$99]. Some years later, they refined the latter in [ABBL05], whom over the previous period formulated a *principle of rejection* (also *causal-rejection principle* [ALP$^+$99, EFST02, ABBL05]) in the above citations.

Informally, the *refined principle* consists of rejecting rules of previous and up-coming programs in an update sequence whenever there are other rules at the current state with which they *conflict*.

Starting with motivation in [ABBL05], they claim to give a simple example to what they called a *tautology* (a rule from which they expect no extra models):

$$\text{not } p \leftarrow \text{not } p \tag{18}$$

Of course that rule alone does not produce any new model in their semantics —i.e. just the **empty model**, $\{\}$. However, it is a clear counterexample why *strong negation* in our framework should not be a simple replacement to *"*$\text{not } \ell$*" in heads*.

Example 4. *Take for example, the program*

$$\{\neg p \leftarrow \text{not } p\} \tag{19}$$

whose unique ***answer set*** *is not the empty set. Namely, the answer set of (19) is just* $\{\neg p\}$.

What is more, in their article, [ABBL05] explain in a footnote what *tautology* means: A rule of the form $\ell \leftarrow \mathsf{Body}$ with $\ell \in \mathsf{Body}$, where ℓ and Body are an atom (or *default-negated atom*) and the body of a rule, respectively. This high dependency in syntax will prove to be one of their major shortcomings, as explained along this thesis.

Before starting with their proper definitions, a very special notation, taken from [ABBL05], is necessary.

Let $\mathcal{A}$ be a set of propositional atoms. As before introduced, a *default literal* is an atom preceded by "not", while a *literal* is either an atom or a default literal. A *rule* ρ is an ordered pair $\mathsf{Head}(\rho) \leftarrow \mathsf{Body}(\rho)$ where $\mathsf{Head}(\rho)$ (the head of the rule) is a literal and $\mathsf{Body}(\rho)$ is a finite set of literals, and it has the form $\mathbf{L}_0 \leftarrow \mathbf{L}_1, \ldots, \mathbf{L}_n$. A rule with $\mathsf{Head}(\rho) = \mathbf{L}_0$ and $\mathsf{Body}(\rho) = \emptyset$ is called a fact, simply written as $\mathbf{L}_0$.

A *generalised logic program* (GLP) Π over $\mathcal{A}$, is a finite or infinite set of rules, and $\Pi_\emptyset$ denotes an empty set of rules. If $\mathsf{Head}(\rho) = a$ (resp. $\mathsf{Head}(\rho) = \mathrm{not}\ a$) then $\mathrm{not}\ \mathsf{Head}(\rho) = \mathrm{not}\ a$ (resp. $\mathrm{not}\ \mathsf{Head}(\rho) = a$). Two rules ρ and ρ' are in *conflict*, denoted by themselves as $\rho \bowtie \rho'$, if and only if $\mathsf{Head}(\rho) = \mathrm{not}\ \mathsf{Head}(\rho')$. An interpretation M of $\mathcal{A}$ is a set of atoms such that $\mathsf{M} \subseteq \mathcal{A}$. An atom a is true in M, denoted by $\mathsf{M} \models a$, if and only if $a \in \mathsf{M}$, and false otherwise. A default literal $\mathrm{not}\ a$ is true in M, denoted by $\mathsf{M} \models \mathrm{not}\ a$, if and only if $a \notin \mathsf{M}$, and false otherwise. A set of literals $\mathbf{L}$ is true in M, denoted by $\mathsf{M} \models \mathbf{L}$, if and only if each literal in $\mathbf{L}$ is true in M. A rule ρ is satisfied by an interpretation M if and only if whenever $\mathsf{M} \models \mathsf{Body}(\rho)$ then $\mathsf{M} \models \mathsf{Head}(\rho)$. An interpretation M is a model of a program Π if and only if M satisfies all rules in Π. An interpretation M of $\mathcal{A}$ is a stable model of a generalised logic program Π if and only if

$$\overline{\mathsf{M}} = \mathsf{least}(\Pi \cup \{not\_a \mid a \notin \mathsf{M}\}) \tag{20}$$

where[3]

$$\overline{\mathsf{M}} = \mathsf{M} \cup \{not\_a \mid a \notin \mathsf{M}\}$$

with a as an atom and with $\mathsf{least}(\cdot)$ as the *least model* of the definite program obtained from the argument program by replacing every default literal $\mathrm{not}\ a$ by a new atom not_a[4].

With this notation, one can define a *dynamic program* as follows.

Definition 3 (Dynamic Logic Program, DyLP [ABBL05])**.** *A* dynamic logic program *(DyLP) is a sequence of generalised logic programs. Let* $\mathcal{P} = (\Pi_1, \ldots, \Pi_s)$ *and* $\mathcal{P}' = (\Pi'_1, \ldots, \Pi'_s)$ *be two DyLP's. The expression* $\rho(\Pi)$ *denotes the* set of all rules appearing in the programs $\Pi_1, \ldots, \Pi_s$*, and* $\mathcal{P} \cup \mathcal{P}'$ *denotes the DyLP:* $(\Pi_1 \cup \Pi'_1, \ldots, \Pi_s \cup \Pi'_s)$.

Lastly, the *refined interpretation of a DyLP* program consists in computing the least model of the positive program that results from the difference of the rejected rules, where the intuition behind $\mathsf{Rej}(\cdot, \cdot)$ is the set of rules that are in conflict with both

[3] Consider that there seems to be a typo in [ABBL05]: they typed "$\mathrm{not}\ a$" rather than "not_a" in (20).

[4] Note that indeed this sort of atoms is positive.

current and previous rules in the sequence. Moreover, $\mathsf{Def}(\cdot,\cdot)$ consists of the positive "default-negated" atoms that do not appear in the intended model.

Definition 4 (Dynamic Stable Model [ABBL05])**.** *Let $\mathcal{P}$ be a dynamic logic program and M an interpretation. M is a* dynamic stable model *of $\mathcal{P}$ if and only if*

$$\overline{M} = \mathit{least}(\rho(\mathcal{P}) \setminus \mathit{Rej}(\mathcal{P}, M) \cup \mathit{Def}(\mathcal{P}, M)) \tag{21}$$

where

$$\mathit{Rej}(\mathcal{P}, M) = \{\rho \mid \rho \in \Pi_i, \exists \rho' \in \Pi_j, i \leq j, \rho \bowtie \rho', M \models \mathit{Body}(\rho')\} \tag{22}$$

and[5]

$$\mathit{Def}(\mathcal{P}, M) = \{not\_a \mid \nexists \rho \in \rho(\mathcal{P}), \mathit{Head}(\rho) = a, M \models \mathit{Body}(\rho)\} \tag{23}$$

$\oplus_R$ is the corresponding update operator.

This approach has had several implementations for download, including one for the original version before the refined principle, and another for the *refined principle*. LUPS is also implemented and the following list shows their respective locations:

- `centria.di.fct.unl.pt/~jja/updates/dlp.html`
- `centria.di.fct.unl.pt/~banti/FedericoBantiHomepage/refdlp.htm`
- `centria.di.fct.unl.pt/~jja/updates/lups.html`

By considering Example 3 again, and inspired from the original example from [ABBL05], the reader may rewrite the pair of programs as follows.

Example 5. *Let $\Pi_1 \oplus_R \Pi_2$, where*

$$\begin{aligned} \Pi_1 = \{ day &\leftarrow \text{not } night \\ night &\leftarrow \text{not } day \\ stars &\leftarrow night, \text{ not } cloudy \\ not\_stars\} & \end{aligned}$$

and

$$\begin{aligned} \Pi_2 = \{ stars &\leftarrow constellations \\ constellations &\leftarrow stars\} \end{aligned}$$

***One of the resulting** dynamic stable models is just $M = \{day\}$ because*

$$\overline{M} = \{day, not\_night, not\_cloudy, not\_stars, not\_constellations\}$$

[5] Notice that it seems they missed the "_" under their "not α" in (23).

where

$$\mathit{Rej}(\mathcal{P}, M) = \{stars \leftarrow night,\ \text{not}\ cloudy\}$$

and

$$\mathit{Def}(\mathcal{P}, M) = \{not\_night, not\_stars, not\_cloudy, not\_constellations\}$$

Thus, $\overline{M} = M \cup \{not\_night, not\_stars, not\_cloudy, not\_constellations\}$. *However, the interpretation* $M = \{night, stars, constellations\}$ ***is also*** *a (refined) dynamic stable model, simply because* $\mathit{Def}(\mathcal{P}, M) = \{not\_cloudy, not\_day\}$ *and* $\mathit{Rej}(\mathcal{P}, M) = \{not\_stars\}$. *As a result, one of the* ***least*** *models is*

$$\overline{M} = \{night, stars, constellations\} \cup \mathit{Def}(\mathcal{P}, M) = M \cup \mathit{Def}(\mathcal{P}, M)$$

which is a clear disadvantage, besides the unnecessary emulation of strong negation and default negation in heads.

Although it is true that [ABBL05] were some of the first people to formulate and implement a semantics for updates, one can easily realise the clear shortcomings this approach has: firstly for the different syntax of the so called generalised logic programs that is a different case of SM—a *non-standard concept of SM* [EFST02]; secondly for the principle itself that produces *counterintuitive results*.

4 Sakama & Inoue —SI

According to the authors in [SI99, SI03], there exist three types of updates: *inconsistency removal*, *view updates* and *theory updates*. Each of those types is a special case of updates and revision.

In particular, the present thesis focuses on theory updates, rather than other special cases of making an inconsistent program consistent or differentiating between variant and invariant knowledge. As a result, I do not go through the other types here, although they are sometimes are related to the particular problem addressed in this thesis.

Before introducing the definitions for theory updates, some new notation and specialised terminology is necessary to understand their approach. For instance, the authors define their particular framework of abduction, and they call it *Extended Abduction*. This framework differs from the standard abduction definition in [Poo88, KM90, KKT98]. For instance, besides an explanation to satisfy

$$K \cup E \models G$$

they also introduce the notion of *negative explanations*, such that

$$K \setminus F \models G$$

where K is a first-order theory; E, F sets of hypotheses; G an observation; and $K \cup E$ and $K \setminus F$ are consistent.

According to [SI03], an *extended abductive program* is a pair $\langle P, \mathcal{A}^* \rangle$ where P and $\mathcal{A}^*$ are DLP's. An *abductive program* $\langle P, \mathcal{A}^* \rangle$ is *consistent* if P is consistent.

In the process of updating a program with another, the authors define a set of conditions that the intended updated must meet.

Definition 5 (Theory Updates [SI03])**.** *Given a DLP-program pair P and Q, P' accomplishes a* theory update *of P by Q if*

1. *P' is consistent,*
2. *$Q \subseteq P' \subseteq P \cup Q$,*
3. *there is no consistent program P'' such that $P' \subset P'' \subseteq P \cup Q$.*

In words of [SI03], the intended update is the union of the new information and a maximal subset of the original program that is consistent with the update, which obviously is not always unique.

Their update process starts as an extended abductive program

$$\langle P \cup Q, P \setminus Q \rangle$$

The intuition behind this program consists of merging the update with the original theory and combining the rules of P that do not belong to Q so as to get a consistent update.

In order to reduce the set of abducible rules $P \setminus Q$ to abductive facts and to compute their models in a conventional way, the extended abductive program (as defined in [IS95]) must be transformed into a *normal* (traditional) abductive program —like in [KKT98]— in which abducibles contain only non-disjunctive facts, as in the bellow definition. In this manner, the models of an update program (later introduced) will contain both facts and names of rules to remove, rather than the rules themselves.

Definition 6 (Normalised Abductive Program [SI03])**.** *Given an extended abductive program $\langle P, \mathcal{A}^* \rangle$, and*

$$\mathcal{R} = \{\Sigma \leftarrow \Gamma \mid (\Sigma \leftarrow \Gamma) \in \mathcal{A}^* \text{ and } \Sigma \leftarrow \Gamma \text{ is not a non-disjunctive fact}\}.$$

Then, let

$$\begin{aligned} P^{\mathrm{n}} &= (P \setminus \mathcal{R}) \cup \{\Sigma \leftarrow \Gamma, \gamma_R \mid R = (\Sigma \leftarrow \Gamma) \in \mathcal{R}\} \\ &\quad \cup \{\gamma_R \leftarrow \mid R \in \mathcal{R} \cap P\}, \\ \mathcal{A}^{*\mathrm{n}} &= (\mathcal{A}^* \setminus \mathcal{R}) \cup \{\gamma_R \mid R \in \mathcal{R}\}, \end{aligned}$$

where γ_R is a newly introduced atom (called the name of *R) uniquely associated with each rule R in $\mathcal{R}$. For any rule $R \in \mathcal{R}$, its name comes from the function $n(R) = \gamma_R$. In particular, any abducible fact $L \leftarrow \top$ has the name L, i.e., $n(L) = L$.*

Once the extended abductive program is normalised, its interpretation is the models of an update program that consists of the rules of the original theory that are not in the normalised abductive set, merged with a new set of update rules, as following specified.

Definition 7 (Update Rules; Update Atoms [SI03])**.** *Given an extended abductive program* $\langle P, \mathcal{A}^* \rangle$*, where* $\mathcal{A}^*$ *contains only (non-disjunctive) facts, the set* UR *of* update rules *is constructed as follows.*

1. *For any literal* $a \in \mathcal{A}^*$*, the following rules are in* UR*:*

$$a \leftarrow \text{not } \overline{a},$$
$$\overline{a} \leftarrow \text{not } a,$$

   where $\overline{a}$ *is a newly introduced atom uniquely associated with* a*. The above pair of rules function is* $abd(a)$ *hereafter. In addition, yet another semantically equivalent way to represent it, according to [SI03], is by* $a \vee \overline{a} \leftarrow \top$*.*

2. *For any literal* $a \in \mathcal{A}^* \setminus P$*, the following rule is in* UR*:*

$$a^{\text{ON}} \leftarrow a.$$

3. *For any literal* $a \in \mathcal{A}^* \cap P$*, the following rule is in* UR*:*

$$a^{\text{OUT}} \leftarrow \text{not } a.$$

where a^{ON} *and* a^{OUT} *are atoms uniquely associated with any* $a \in \mathcal{A}^*$*, so called* update atoms*,* UA*.*

[SI03] interpret, at a *meta-level*, that a^{ON} means making a true, when it is not in P, while a^{OUT} means making a not true when it is in P. In other words, they represent the introduction and deletion of a, respectively. On the other hand, $\overline{a}$ would mimic an *unknown truth value* of a, i.e. neither true nor false a. Last, they define the set of all update atoms associated with the abducibles in $\mathcal{A}^*$ by $\mathcal{UA}$. That is to say, $\mathcal{UA} = \mathcal{UA}^{\text{ON}} \cup \mathcal{UA}^{\text{OUT}}$, where $\mathcal{UA}^{\text{ON}}$ (resp. $\mathcal{UA}^{\text{OUT}}$) is the set of update atoms of the form a^{ON} (resp. a^{OUT}).

Next, these update rules take part of the update program of the normalised extended abductive program that is an intermediate DLP. This intermediate program specification is as follows:

Definition 8 (Update Programs, UP [SI03])**.** *Given an extended abductive program* $\langle P, \mathcal{A}^* \rangle$*, its* update program UP *is defined as a* DLP *such that*

$$\mathsf{UP} = (P \setminus \mathcal{A}^*) \cup \mathsf{UR}.$$

Then they compute the models of an update program, which point out the deletion of facts or rules from the original program in the pair. As a result, a new updated program (or more than one) comes up.

Definition 9 (U-minimal Answer Sets [SI03])**.** *An answer set $\mathcal{S}$ of UP is called* U-minimal *(U-MAS) if there is no answer set $\mathcal{S}'$ of UP such that $\mathcal{S}' \cap \mathcal{UA} \subset \mathcal{S} \cap \mathcal{UA}$.*

This abduction framework proofs to have nice properties of minimal change when performing particular kinds of updates of non-monotonic theories, and when maintaining their consistency, with a vast analysis of shortcomings in other approaches. Rather than characterising updates through abduction, however, [SI03]'s first goal is the converse, providing a mechanism (an update semantics) to characterise their extended abduction, as they themselves state. Consequently, the approach lacks of a proper analysis of more principles and postulates from the literature. Additionally, they characterise different kinds of updates with their extended abduction, claiming that they can provide an algebra of rules deletion, besides the addition of them, to explain observations.

Let us carry on by recapitulating their approach. They define a called update program out of the normal abductive form of an extended abductive program

$$\langle\, P \cup Q, P \setminus Q \,\rangle$$

whose models are *U-MAS's*, interpreted from an update program. Last, the interpretation produces one (or more) new programs representing knowledge bases, derived from the addition/deletion of facts that the U-MAS's describe in turn.

In order to illustrate the above definitions, consider the following example extended from the original in [SI03][6] that shows one of the differences with several approaches.

Example 6. *Suppose an update to the knowledge base*

$$\begin{aligned} \Pi_1 = \{ sleep &\leftarrow \text{not } tvon \\ watchtv &\leftarrow tvon \\ tvon &\leftarrow \top \} \end{aligned}$$

with[7]

$$\begin{aligned} \Pi_2 = \{ powerfailure &\leftarrow \top \\ \bot &\leftarrow powerfailure, tvon \} \end{aligned}$$

The situation is in the abductive program $\langle\, \Pi_1 \cup \Pi_2, \Pi_1 \setminus \Pi_2 \,\rangle$. The update program

[6] Originally, [ALP$^+$99] proposed this example, but it is a little modified in [SI03] to contrast their differences. Moreover, I have extended it here in order to see further details.

[7] In [ALP$^+$99] the rule "$\leftarrow powerfailure, tvon$" is given as "not $tvon \leftarrow powerfailure$". These two rules are semantically equivalent under the answer set semantics, as the authors explain in [SI03]. However, as later seen in this example, the difference between either expression would result in the existence of $\neg tvon$ in the corresponding model!

UP of $\langle\, (\Pi_1 \cup \Pi_2)^{\mathrm{n}}, (\Pi_1 \setminus \Pi_2)^{\mathrm{n}} \,\rangle$ *is specified as*

$$\begin{aligned} \mathit{UP}: \quad & \mathit{powerfailure} \leftarrow \top \\ & \bot \leftarrow \mathit{powerfailure}, \mathit{tvon} \\ & \mathit{sleep} \leftarrow \mathrm{not}\ \mathit{tvon}, \gamma_1 \\ & \mathit{watchtv} \leftarrow \mathit{tvon}, \gamma_2 \\ & \mathit{abd}(\mathit{tvon}), \mathit{abd}(\gamma_1), \mathit{abd}(\gamma_2), \\ & \mathit{tvon}^{\mathrm{OUT}} \leftarrow \mathrm{not}\ \mathit{tvon} \\ & \gamma_1^{\mathrm{OUT}} \leftarrow \mathrm{not}\ \gamma_1 \\ & \gamma_2^{\mathrm{OUT}} \leftarrow \mathrm{not}\ \gamma_2 \end{aligned}$$

where γ_1 *and* γ_2 *are names of the abducible rules in* $\Pi_1 \setminus \Pi_2$. *Then, UP has the unique U-MAS*

$$\{\mathit{powerfailure}, \mathit{sleep}, \overline{\mathit{tvon}}, \mathit{tvon}^{\mathrm{OUT}}, \gamma_1, \gamma_2\}$$

which represents the deletion of fact tvon *from* $\Pi_1 \cup \Pi_2$. *As a result, the theory update of* Π_1 *by* Π_2 *becomes*

$$\begin{aligned} \Pi_3: \quad & \mathit{sleep} \leftarrow \mathrm{not}\ \mathit{tvon} \\ & \mathit{watchtv} \leftarrow \mathit{tvon} \\ & \mathit{powerfailure} \leftarrow \top \\ & \bot \leftarrow \mathit{powerfailure}, \mathit{tvon} \end{aligned}$$

whose ***answer set*** *is just* $\{\mathit{powerfailure}, \mathit{sleep}\}$

Next, suppose ***yet another update***

$$\Pi_4 : \neg \mathit{powerfailure} \leftarrow \top$$

to Π_3, *which represents that the power is back again. [SI03] code this new pair by the abductive program* $\langle\, \Pi_3 \cup \Pi_4, \Pi_3 \setminus \Pi_4 \,\rangle$. *So, the update program of* $\langle\, (\Pi_3 \cup \Pi_4)^{\mathrm{n}}, (\Pi_3 \setminus \Pi_4)^{\mathrm{n}} \,\rangle$ *turns into*

$$\begin{aligned} \mathit{UP}: \quad & \neg \mathit{powerfailure} \leftarrow \top \\ & \mathit{sleep} \leftarrow \mathrm{not}\ \mathit{tvon}\gamma_1 \\ & \mathit{watchtv} \leftarrow \mathit{tvon}\gamma_2 \\ & \bot \leftarrow \mathit{powerfailure}, \mathit{tvon}\gamma_3 \\ & \mathit{abd}(\mathit{powerfailure}), \mathit{abd}(\gamma_1), \mathit{abd}(\gamma_2), \mathit{abd}(\gamma_3), \\ & \mathit{powerfailure}^{\mathrm{OUT}} \leftarrow \mathrm{not}\ \mathit{powerfailure} \\ & \neg\gamma_1 \leftarrow \mathrm{not}\ \gamma_1, \neg\gamma_2 \leftarrow \mathrm{not}\ \gamma_2, \neg\gamma_3 \leftarrow \mathrm{not}\ \gamma_3. \end{aligned}$$

Then, UP has the unique U-MAS

$$\{\neg \mathit{powerfailure}, \mathit{sleep}, \gamma_1, \gamma_2, \gamma_3, \overline{\mathit{powerfailure}}, \neg \mathit{powerfailure}\}$$

which implies that the result of the update is

$$(\Pi_3 \cup \Pi_4) \setminus \{powerfailure \leftarrow \top\}$$

As a result, the ***unique answer set*** *of the unique resulting program is*

$$\{\neg powerfailure, sleep\}$$

[SI03] propose this example as an argument against other approaches like [ALP+99, EFST02], that bring back previous knowledge of the original theory. That is to say, their interpretation is that the TV turns itself on again and it is possible to watch it as well: $\{tvon, watchtv, \neg powerfailure\}$, which does not coincide with their intuition. However, this argument seems to be too strong to generalise that all update semantics should behave accordingly, because the authors are differentiating *fluents* and *actions* in a language that does not have such an explicit difference.

In order to illustrate this assumption, let us modify Example 6 in such a way that the language contains only fluents, naturally at a higher abstraction level. Therefore, the new story goes like this.

Example 7. *Suppose a learning agent whose simple knowledge base states that it is innocent when it is not guilty, and at the beginning it believes it is not guilty, thus innocent. A following update states that the agent is no longer innocent when guilty, that it is guilty when a murderer and now it is a murderer. Thus, it is no longer innocent. However, more relevant rules pop up that state that an agent is not a murderer when self defended; that is self defended when first attacked; and it is a fact that the agent was first attacked. Consequently, common intuition would dictate that the agent's innocence should be in effect. Under [SI03]'s approach, however, that previous knowledge is lost forever and there is no way to conclude that it is innocent.*

Here is how [SI03]'s approach represents this knowledge. The knowledge base consists of an original theory, Π_1*, as well as an update to it,* Π_2*, where*

$$\begin{aligned} \Pi_1 : \{innocent &\leftarrow \neg guilty \\ \neg guilty &\leftarrow \top\} \\ \Pi_2 : \{\neg innocent &\leftarrow guilty \\ guilty &\leftarrow murderer \\ murderer &\leftarrow \top\} \end{aligned}$$

and its normalised abductive program $\langle P^n, \mathcal{A}^{*n} \rangle$ *where*

$$\begin{aligned} P^n = \{\neg guilty &\leftarrow \top \\ \gamma_1 &\leftarrow \top \\ murderer &\leftarrow \top \\ innocent &\leftarrow \neg guilty, \gamma_1 \\ \neg innocent &\leftarrow guilty \\ guilty &\leftarrow murderer\} \\ \mathcal{A}^{*n} = \{\neg guilty &\leftarrow \top \\ \gamma_1 &\leftarrow \top\} \end{aligned}$$

The update rules and the update program consist respectively of

$$\begin{aligned} UR : \{\neg guilty \vee \overline{\neg guilty} &\leftarrow \top \\ \neg guilty^{\mathrm{OUT}} &\leftarrow \mathrm{not}\ \neg guilty \\ \gamma_1 \vee \overline{\gamma_1} &\leftarrow \top \\ \gamma_1^{\mathrm{OUT}} &\leftarrow \mathrm{not}\ \gamma_1\} \\ \mathit{UP} : \{\neg guilty \vee \overline{\neg guilty} &\leftarrow \top \\ \neg guilty^{\mathrm{OUT}} &\leftarrow \mathrm{not}\ \neg guilty \\ \gamma_1 \vee \overline{\gamma_1} &\leftarrow \top \\ \gamma_1^{\mathrm{OUT}} &\leftarrow \mathrm{not}\ \gamma_1 \\ murderer &\leftarrow \top \\ innocent &\leftarrow \neg guilty, \gamma_1 \\ \neg innocent &\leftarrow guilty \\ guilty &\leftarrow murderer\} \end{aligned}$$

that has two answer sets

$$\{murderer, \overline{\neg guilty}, \neg guilty^{\mathrm{OUT}}, \overline{\gamma_1}, \gamma_1^{\mathrm{OUT}}, \neg innocent, guilty\}$$
$$\{murderer, \overline{\neg guilty}, \neg guilty^{\mathrm{OUT}}, \gamma_1, \neg innocent, guilty\}$$

from which the unique U-MAS

$$\{murderer, \overline{\neg guilty}, \neg guilty^{\mathrm{OUT}}, \gamma_1, \neg innocent, guilty\}$$

leads to the ***updated knowledge base*** *where* $\neg guilty$ *is no longer present:*

$$\begin{aligned} \Pi_3 : \{innocent &\leftarrow \neg guilty \\ \neg innocent &\leftarrow guilty \\ guilty &\leftarrow murderer \\ murderer &\leftarrow \top\} \end{aligned}$$

Next, the following program represents the second update as:

$$\begin{array}{rcl} P_4 : \{\neg murderer & \leftarrow & self\_defence \\ self\_defence & \leftarrow & attacked \\ attacked & \leftarrow & \top\} \end{array}$$

which, after the same process yields the following update program

$$\begin{array}{rcl} \mathit{UP} : \{\gamma_1 \vee \overline{\gamma_1} & \leftarrow & \top \\ \gamma_1^{\mathrm{OUT}} & \leftarrow & \mathrm{not}\ \gamma_1 \\ \gamma_2 \vee \overline{\gamma_2} & \leftarrow & \top \\ \gamma_2^{\mathrm{OUT}} & \leftarrow & \mathrm{not}\ \gamma_2 \\ murderer \vee \overline{murderer} & \leftarrow & \top \\ murderer^{\mathrm{OUT}} & \leftarrow & \mathrm{not}\ murderer \\ \gamma_3 \vee \overline{\gamma_3} & \leftarrow & \top \\ \gamma_3^{\mathrm{OUT}} & \leftarrow & \mathrm{not}\ \gamma_3 \\ attacked & \leftarrow & \top \\ innocent & \leftarrow & \neg guilty, \gamma_1 \\ \neg innocent & \leftarrow & guilty, \gamma_2 \\ guilty & \leftarrow & murderer, \gamma_3 \\ \neg murderer & \leftarrow & self\_defence \\ self\_defence & \leftarrow & attacked\} \end{array}$$

with the unique U-MAS

$$\{attacked, \gamma_1, \gamma_2, \overline{murderer}, murderer^{\mathrm{OUT}}, \gamma_3, \neg murderer, self\_defence\}$$

that produces a knowledge base

$$\begin{array}{rcl} P_5 : \{innocent & \leftarrow & \neg guilty \\ \neg innocent & \leftarrow & guilty \\ guilty & \leftarrow & murderer \\ \neg murderer & \leftarrow & self\_defence \\ self\_defence & \leftarrow & attacked \\ attacked & \leftarrow & \top\} \end{array}$$

that ***models*** $\{attacked, \neg murderer, self\_defence\}$ *reflects the loss of previous relevant information —no conclusions about guilt or innocence are available.*

If this *counterintuitive example* was not enough, let us change a bit the original Example 6 in such a way that both actions and fluents are inverted.

Example 8. *Suppose a simple scenario in where an agent can see in a room where its blinds are open. Later, new information is at hand and the agent knows that it cannot see when the blinds are closed, that by closing them means they are closed, and that they cannot be closed and open at the same time. Simultaneously, there is also an event of closing the blinds. Following, a program that codes the initial information:*

$$\begin{aligned}\Pi_1 : \{ \mathit{can\_see} &\leftarrow \mathit{blinds\_open} \\ \mathit{blinds\_open} &\leftarrow \top\}\end{aligned}$$

updated with

$$\begin{aligned}\Pi_2 : \{\neg \mathit{can\_see} &\leftarrow \mathit{blinds\_closed} \\ \bot &\leftarrow \mathit{blinds\_open}, \mathit{blinds\_closed} \\ \mathit{blinds\_closed} &\leftarrow \mathit{close\_blinds} \\ \mathit{close\_blinds} &\leftarrow \top\}\end{aligned}$$

After updating Π_1 *with* Π_2*, the update program*

$$\begin{aligned}\mathit{UP} : \{\mathit{blinds\_open} \vee \overline{\mathit{blinds\_open}} &\leftarrow \top \\ \mathit{blinds\_open}^{\mathrm{OUT}} &\leftarrow \mathrm{not}\ \mathit{blinds\_open} \\ \gamma_1 \vee \overline{\gamma_1} &\leftarrow \top \\ \gamma_1^{\mathrm{OUT}} &\leftarrow \mathrm{not}\ \gamma_1 \\ \mathit{close\_blinds} &\leftarrow \top \\ \mathit{can\_see} &\leftarrow \mathit{blinds\_open}, \gamma_1 \\ \neg \mathit{can\_see} &\leftarrow \mathit{blinds\_closed} \\ \bot &\leftarrow \mathit{blinds\_open}, \mathit{blinds\_closed} \\ \mathit{blinds\_closed} &\leftarrow \mathit{close\_blinds}\}\end{aligned}$$

has the following U-MAS:

$$\{\mathit{close\_blinds}, \overline{\mathit{blinds\_open}}, \mathit{blinds\_open}^{\mathrm{OUT}}, \gamma_1, \neg \mathit{can\_see}, \mathit{blinds\_closed}\}$$

This model means the deletion of fact $\mathit{blinds\_open}$ *from the original knowledge base.*

Now suppose the agent decides not to close the blinds when it is reading, that it is reading when it wants to read, and that now it wants to read. Then, the updated

program and the new update are

$$\begin{aligned}
\{can_see &\leftarrow blinds_open\\
\neg can_see &\leftarrow blinds_closed\\
\bot &\leftarrow blinds_open, blinds_closed\\
blinds_closed &\leftarrow close_blinds\\
close_blinds &\leftarrow \top\}\\
\{\neg close_blinds &\leftarrow reading\\
reading &\leftarrow want_to_read\\
want_to_read &\leftarrow \top\}
\end{aligned}$$

whose unique U-MAS

$$\{want\_to\_read, \gamma_1, \gamma_2, \gamma_3, \overline{close\_blinds}, close\_blinds^{\mathrm{OUT}}, \gamma_4, \neg close\_blinds, reading\}$$

produces an updated program

$$\begin{aligned}
\{can_see &\leftarrow blinds_open\\
\neg can_see &\leftarrow blinds_closed\\
\bot &\leftarrow blinds_open, blinds_closed\\
blinds_closed &\leftarrow close_blinds\\
\neg close_blinds &\leftarrow reading\\
reading &\leftarrow want_to_read\\
want_to_read &\leftarrow \top\}
\end{aligned}$$

with an answer set that again reflects a loss of information on the ability to see:

$$\{want\_to\_read, \neg close\_blinds, reading\}$$

Clearly, the objection [SI03] propose against other semantics may have different interpretations in *planning scenarios*, where there is indeed a formal explicit distinction between *fluents and actions*. Meanwhile, neither interpretation is correct or incorrect when talking about simple logic-program updates, unless formalising which rules must persist and which must not.

Moreover, although the authors present a deep analysis of their proposal and although it seems to be *robust*-enough for agent's *changing environment*, there is a lack of further and more *general properties* that makes it hard to compare with other competitors.

5 Zhang's line

An interesting proposal for updates comes from [Zha06], where the author identifies three types of problems to solve in an update process: *elimination of contradictory*

information, *conflict resolution* and *syntactic representation*.

Additionally, one of the applications from that line is an interesting language introduced in [CZ05] that is specialised in updates of agent *policies* and defined at the top of ASP. [CZ05] specify such policies in terms of clauses with a predefined semi-imperative *syntactical structure*, as well as an initial *planning approach*.

However, owing to a special focus the work has on *policies*, the programmer is restricted and obliged to use *reserved words* like "always", "implied by", "with absence", etc. which, besides constraining the domain to specific applications, it 'reduces' the language and has potentially different meanings in the *meta-language*. Nevertheless, they already have a fully-fledged system, as they themselves mention it in [CZ05].

5.1 General View

As mentioned above, [Zha06] characterises updates in terms of three main objectives: *contradiction elimination*, *conflict resolution* and *syntactic representation*. The first topic is one of the most obvious in semantics for updates, which should be real by preserving a *minimal-change principle* and a proper justification. On the other hand, *conflict resolution* has to do with potential future *contradictions* an update might yield because of the introduction of the two kinds of negations in logic programs —strong and default negation. Finally, once a semantics meets the two main goals, the author argues that a proper semantics should also *preserve* as many as possible of the original rules from the *updating knowledge base*.

In order to realise these three goals, [Zha03] characterises a program update by means of a called *prioritised logic program*. In an intuitive way, this kind of program consists in *preferring the latest update* to the *original knowledge base*.

[Zha06] motivates his proposal by introducing a clever example that exposes the two kinds of problems he studied, and the example I borrow looks as follows:

Example 9 ([ZF05])**.** *Suppose*

$$\begin{aligned}
\Pi_0 = \{ member(a,g) &\leftarrow \top && (24)\\
member(b,g) &\leftarrow \top && (25)\\
access(a,f_2) &\leftarrow \top && (26)\\
access(X,f_1) &\leftarrow member(X,g) && (27)\\
\neg access(X,f_2) &\leftarrow member(X,g), \text{not } access(X,f_2)\} && (28)
\end{aligned}$$

updated with

$$\begin{aligned}
\Pi_1 = \{ member(c,g) &\leftarrow \top && (29)\\
\neg access(X,f_1) &\leftarrow member(X,g) && (30)\\
access(X,f_2) &\leftarrow member(X,g), \text{not } \neg access(X,f_2)\} && (31)
\end{aligned}$$

According to [Zha06], this update ought to have the unique answer set

$$S = \{\neg access(a, f_1), \neg access(b, f_1), \neg access(c, f_1), \\ access(a, f_2), access(b, f_2), access(c, f_2)\}$$

Then he claims that rule (31) should *override* rule (28)! That is to say, [Zha06] states that there is information loss in some other semantics, but at the same time, his semantics says both b and c have access to f_2, ignoring the possible situation (world) when they explicitly do not. In fact, one might expect that b has no access to f_2 in Π_0 and that such a situation persists.

Regarding the controversy from this syntactical change of rule, his approach proposes a two-fold process of *eliminating contradictory information*, as well as *resolution of conflicting rules* and a final *syntactic representation* stage. Nevertheless, before the introduction of those two main processes, some fundamental definitions are in order.

The following definition can be seen as assuming true the given ground literals in $\mathcal{S}$ to Π:

Definition 10 (e-program, $e(\Pi, \mathcal{S})$ [Zha06])**.** *Given a set of ground literals $\mathcal{S}$, $e(\Pi, \mathcal{S})$ denotes the program obtained from program Π by deleting*

1. *each rule in Π that has a formula $\text{not } \ell$ in its body with $\ell \in \mathcal{S}$, and*
2. *all formulas of form ℓ in the bodies of the remaining rules with $\ell \in \mathcal{S}$.*

The following example from [Zha06] illustrates the definition.

Example 10 ([Zha06])**.** *Given $\mathcal{S} = \{a, \neg b\}$ and the program*

$$\Pi = \{c \leftarrow a \\ \neg d \leftarrow \text{not } a\}$$

$e(\Pi, S) = \{c \leftarrow \top\}$.

This definition will prove useful to test a coherence concept in [Zha06]'s approach.

Definition 11 (Coherence [Zha06])**.** *A set of ground literals $\mathcal{S}$ is* coherent *with an extended logic program Π if for any answer set $\mathcal{S}'$ of $e(\Pi, \mathcal{S})$, $\mathcal{S} \cup \mathcal{S}'$ is consistent.*

By continuing Example 10, the only answer set of $e(\Pi, S)$ is $\{c\}$. Thus, $\mathcal{S}$ is coherent with Π.

As another example, let us consider [Zha06]'s: $\mathcal{S} = \{a, \neg b\}$ is coherent with $\Pi = \{c \leftarrow a, \neg d \leftarrow \text{not } a\}$ because the only answer set of $e(\Pi, \mathcal{S})$ is $\{c\}$. However, $\{a, \neg b, \neg c\}$ is not coherent with Π.

These are basic steps towards a general proposal that consists in *two* main steps to perform an update of two programs. Firstly, *eliminating contradictory rules* from a previous program with respect to the latest one. Secondly, the semantics solves *conflicts* between the remaining rules of the programs. The semantics that determines the specifications of such an elimination and conflict resolution is *Prioritised Logic Programs*.

5.2 Prioritised Logic Programs

In order to specify the algebra for this logic program update proposal, [Zha06] employs an earlier platform called *Prioritised Logic Programming* [ZF97, Zha03], or simply PLP. Informally, this sort of logic programs consists of a set of preference relations and of a naming function that assigns a name to each rule.

Definition 12 (Prioritized Logic Program PLP [Zha06])**.** *A* prioritized logic program $\mathcal{P}$ *is a triple* $(\Pi, \mathcal{N}, <)$*, where* Π *is an extended logic program,* $\mathcal{N}$ *is a naming function mapping each rule in* Π *to a name, and "*$<$*" is a strict partial order on names. Moreover,* $\mathcal{P}(<)$ *denotes the set of* $<$*-relations of* $\mathcal{P}$*.*

According to [Zha06], if $\mathcal{N}(\rho) < \mathcal{N}(\rho')$ holds in $\mathcal{P}$, rule ρ is preferred to be applied over rule ρ' while evaluating $\mathcal{P}$. What is "evaluation" of $\mathcal{P}$ anyhow? The following definitions code what an evaluation is. Meanwhile, it is worth recalling what an extended logic program is, before going any further:

Definition 13 (Extended Logic Program, ELP)**.** *An* extended logic program *is a set of rules of the form*

$$p_0 \leftarrow q_1, \ldots, q_m, \text{not } q_{m+1}, \ldots, \text{not } q_n \tag{32}$$

where p_i *and* q_i *are literals and* $m, n \in \mathbb{N}$*.*

Finally, the definition of a *defeated rule* looks as follows.

Definition 14 (Defeated rule [Zha03])**.** *Let* Π *be a ground extended logic program and* ρ *a ground rule of form (32) —*ρ *does not necessarily belong to* Π*. Rule* ρ *is* defeated *by* Π *if and only if* Π *has an answer set and for any answer set* $\mathcal{S}$ *of* Π*, there exists some* $\ell_i \in \mathcal{S}$*, where* $m + 1 \leq i \leq n$*.*

For example, given a program

$$\begin{aligned} \Pi = \{a &\leftarrow \top \\ c &\leftarrow b \\ d &\leftarrow \text{not } e\} \end{aligned}$$

whose answer set is $\{a, d\}$, Π defeats rules like

$$\begin{aligned} \bot &\leftarrow \text{not } d; \\ a &\leftarrow \text{not } a; \\ c &\leftarrow b, \text{not } a, \text{not } d, \text{not } e \end{aligned}$$

Similarly to the case of extended logic programs, the evaluation of a PLP shall be on its ground form. Moreover, [Zha06] states that a PLP like $\mathcal{P}' = (\Pi', \mathcal{N}', <')$ is the *ground instantiation* of $\mathcal{P} = (\Pi, \mathcal{N}, <)$ if (1) Π' is the ground instantiation of Π;

and (2) $<'$ is a strict partial ordering and $\mathcal{N}'(\rho'_1) <' \mathcal{N}'(\rho'_2) \in \mathcal{P}'(<')$ if and only if there exist rules ρ_1 and ρ_2 in Π such that ρ'_1 and ρ'_2 are ground instances of ρ_1 and ρ_2, respectively, and $\mathcal{N}(\rho_1) < \mathcal{N}(\rho_2) \in \mathcal{P}(<)$.

Definition 15 (Reduct $\mathcal{P}^<$ [Zha03])**.** *Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$ be a prioritized logic program. $\mathcal{P}^<$ is a* reduct of $\mathcal{P}$ *with respect to "$<$" if and only if there exists a sequence of sets Π_i ($i = 0, 1, \dots$) such that:*

1. $\Pi_0 = \Pi$;
2. $\Pi_i = \Pi_{i-1} \setminus \{\rho_1, \rho_2, \dots\}$ *such that the following two conditions hold:*
   (a) *there exists $\rho \in \Pi_{i-1}$ such that for every j ($j = 1, 2, \dots$),*
   $\mathcal{N}(\rho) < \mathcal{N}(\rho_j) \in \mathcal{P}(<)$
   and $\rho_1, \rho_2, \dots$ are defeated by $\Pi_{i-1} \setminus \{\rho_1, \rho_2, \dots\}$
   (b) *there are no rules $\rho', \rho'', \cdots \in \Pi_{i-1}$ such that*
   $N(\rho_j) < N(\rho'), N(\rho_j) < N(\rho''), \dots$
   for some j ($j = 1, 2, \dots$) and $\rho', \rho'', \dots$ are defeated by $\Pi_{i-1} \setminus \{\rho', \rho'', \dots\}$
3. $\mathcal{P}^< = \bigcap_{i=0}^{\infty} \Pi_i$.

In Definition 15, $\mathcal{P}^<$ is an extended logic program that comes from Π by removing some defeated rules from Π, by following the order relations in $\mathcal{P}(<)$ on rules named by $\mathcal{N}$. Specifically, if $\mathcal{N}(\rho) < \mathcal{N}(\rho_1), \mathcal{N}(\rho) < \mathcal{N}(\rho_2), \dots$, and $\Pi_{i-1} \setminus \{\rho_1, \rho_2, \dots\}$ defeats $\{\rho_1, \rho_2, \dots\}$, then, the rules $\rho_1, \rho_2, \dots$ will be out from Π_{i-1} unless *a less preferred rule* than can be removed in turn: conditions (2a) and (2b). One ought to compute the reduct procedure until a fixed point. Note that it is "less preferred" rather than the opposite for, at this stage, there is no update semantics.

In addition, condition (2b) in Definition 15 is necessary. In its absence, some *counterintuitive results* may be derived —[Zha06]. For instance, consider the following example from the same author:

$\mathcal{P}_1 = (\Pi, \mathcal{N}, <)$:
$N_1 : flies(X) \leftarrow bird(X)$, *not* $\neg flies(X)$
$N_2 : \neg flies(X) \leftarrow penguin(X)$, *not* $flies(X)$
$N_3 : bird(tweety) \leftarrow \top$
$N_4 : penguin(tweety) \leftarrow \top$
$N_2 < N_1$

If one added the preference $N_3 < N_2$ in $\mathcal{P}_1$, then using a modified version of Definition 15 without condition (b),

$\{flies(tweety) \leftarrow bird(tweety), \text{not } \neg flies(tweety)$
$bird(tweety) \leftarrow \top$
$penguin(tweety) \leftarrow \top\}$

is a reduct of $\mathcal{P}_1$, from which it concludes that Tweety flies. On the other hand, by considering both the added preference and condition (2b) one will conclude that Tweety does not fly from a unique reduct that lacks rule N_3.

Finally, an interpretation of a PLP is the answer sets of its reduct, as formally specified in the following definition.

Definition 16 (Answer Set of $\mathcal{P}$ [Zha03])**.** *Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$ be a PLP and Lit the set of all ground literals in the language of $\mathcal{P}$. For any subset $\mathcal{S}$ of Lit, $\mathcal{S}$ is an* answer set of $\mathcal{P}$ *if and only if $\mathcal{S}$ is an answer set for some reduct $\mathcal{P}^<$ of $\mathcal{P}$.*

Using Definition 15 and Definition 16, it is easy to conclude that $\mathcal{P}_1$ has a unique reduct as follows:

$$\begin{aligned}\mathcal{P}_1^< = \{&\neg flies(tweety) \leftarrow penguin(tweety), \mathit{not}\ flies(tweety)\\ &bird(tweety) \leftarrow \top\\ &penguin(tweety) \leftarrow \top\}\end{aligned}$$

from which we obtain the following answer set of $\mathcal{P}_1$:

$$\mathcal{S} = \{bird(tweety), penguin(tweety), \neg flies(tweety)\}.$$

Let us analyse a complete example inspired from the same reference [Zha06], which illustrates in detail when a PLP has more than one reduct. Before that, one should be aware that a PLP may or may not have answer sets. In the first case, the program is called *well-defined program*.

Example 11 ([Zha06])**.** *Suppose a PLP consisting of*

$$\begin{aligned}\mathcal{P} = (\Pi, \mathcal{N}, >) := \{N_1 &: a \leftarrow \top\\ N_2 &: b \leftarrow \text{not } c\\ N_3 &: d \leftarrow \top\\ N_4 &: c \leftarrow \text{not } b\\ &\quad N_1 < N_2, N_3 < N_4\}\end{aligned}$$

By Definition 15, one reduct is constructed as

1. $\Pi_0 = \Pi$
2. $\Pi_1 = \Pi_0 \setminus \{b \leftarrow \text{not } c\}$ *because rule $a \leftarrow \top \in \Pi_0$ and with its tag N_1, one finds the relations $N_1 < N_2 \in \mathcal{P}(<)$ and $b \leftarrow \text{not } c$ is defeated by $\Pi_0 \setminus \{b \leftarrow \text{not } c\}$. Last, there are no rules $\rho', \rho'', \cdots \in \Pi_0$, whose tag is "greater than" N_2, and defeated by $\Pi_0 \setminus \{\rho', \rho'', \dots\}$.*
3. *Finally, the reduct is the intersection of the two programs:*

$$\mathcal{P}(<) = \{a \leftarrow \top \quad d \leftarrow \top \quad c \leftarrow \text{not } b\}$$

and the other reduct as

1. $\Pi_0 = \Pi$
2. $\Pi_1 = \Pi_0 \setminus \{c \leftarrow \text{not } b\}$ *because rule* $d \leftarrow \top \in \Pi_0$ *and with its tag* N_3 *there is the relation* $N_3 < N_4 \in \mathcal{P}(<)$ *and* $c \leftarrow \text{not } b$ *is defeated by* $\Pi_0 \setminus \{c \leftarrow \text{not } b\}$ *and there are no rules* $\rho', \rho'', \cdots \in \Pi_0$*, whose tag is "greater than"* N_4*, and defeated by* $\Pi_0 \setminus \{\rho', \rho'', \dots\}$.
3. *Finally,* $\mathcal{P}(<) = \{a \leftarrow \top \quad b \leftarrow \text{not } c \quad d \leftarrow \top\}$

This section of Prioritised Logic Programs is the necessary background to give the interpretation of the following procedure for updates under the approach in [Zha06] that, as mentioned before, it consists in two main steps: *contradiction elimination* and *conflict resolution*.

5.3 Eliminating Contradictions

The first step in updating two extended logic programs under [Zha06]'s approach is eliminating contradictions by means of an extended simple-fact update program. Informally, the extended simple-fact program consist of establishing a high preference to inertia rules over update rules so that facts in the initial knowledge base may persist after an update. Then, it consists in interpreting the semantics of a resulting (possibly empty) update program(s) that should have a minimal difference with the answer sets of the original program. This interpretation of the update program(s) is the same as for PLP's.

Before going straight to the main definition, some minor notation is necessary:

Definition 17 (Initial Knowledge; PLP Languages [Zha06])**.** *It is stated that*

$\mathcal{B}$ *denotes an initial consistent knowledge base of ground literals of a language* $\mathcal{L}$*;*

Π *an update extended logic program over* $\mathcal{L}$*; and*

$\mathcal{L}_{\mathsf{new}}$ *an extension to* $\mathcal{L}$*, by propositional literals of the form* new-$\ell \mid \ell \in \mathcal{L}$.

[Zha06] represents an update program through a triple, and that program specifies the changes to an original knowledge base, according to a new update. Formally,

Definition 18 ($\mathsf{U}_{\mathsf{PLP}}$-specification, $\mathsf{U}_{\mathsf{PLP}}(\mathcal{B}, \Pi)$ [Zha06])**.** *Let* $\mathcal{B}$*,* Π*,* $\mathcal{L}$*, and* $\mathcal{L}_{new}$ *be as above. The* specification of updating $\mathcal{B}$ with Π *is a* PLP *over* $\mathcal{L}_{new}$*, denoted as* $\mathsf{U}_{PLP}(\mathcal{B}, \Pi) = (\Pi^*, \mathcal{N}, <)$*, as follows:*

1. Π^* *consists of following rules:*

   Initial knowledge rules: *for each literal* ℓ *in* $\mathcal{B}$*, there is a rule* $\ell \leftarrow \top$

Inertia rules: *for each predicate symbol*[8]
$P \in \mathcal{L}$, there are two rules:

$$\textsf{new-}P(x) \leftarrow P(x), \text{not } \neg\textsf{new-}P(x)$$

and

$$\neg\textsf{new-}P(x) \leftarrow \neg P(x), \text{not } \textsf{new-}P(x)$$

Update rules: *for each rule*

$$\ell_0 \leftarrow \ell_1, ..., \ell_m, \text{not } \ell_{m+1}, \ldots, \text{not } \ell_n \in \Pi$$

there is a rule[9]

$$\textsf{new-}\ell_0 \leftarrow \textsf{new-}\ell_1, \ldots, \textsf{new-}\ell_m, \text{not } \textsf{new-}\ell_{m+1}, \ldots, \text{not } \textsf{new-}\ell_n$$

2. *Naming function $\mathcal{N}$ assigns a unique name N for each rule in Π^*.*

3. *For any inertia rule ρ and update rule ρ', $\mathcal{N}(\rho) < \mathcal{N}(\rho')$.*

According to Zhang [Zha06], an answer set of Π^* represents a possible resulting knowledge base from the update of $\mathcal{B}$ by Π, and a literal new-ℓ represents the persistence of ℓ if $\ell \in \mathcal{B}$ or a change of ℓ if $\neg\ell \in \mathcal{B}$ or $\ell \notin \mathcal{B}$ with respect to the update. For instance, in Example 12, interpreting Π^* corresponds to simple-fact update semantics, and it yields two answer sets: $\{\neg a, b, c, newa, newc, \neg newb\}$ and $\{\neg a, b, c, newa, newc, newb\}$, which means that the truth value of b is *indefinite* by $newb$ with respect to the update: the new atom $newb$ is true in one answer set and false in the other .

Up to now, one can transform an initial knowledge base into an initial logic program and inertial rules together with the update rules to form a PLP. In addition, one can establish preference relations among PLP rules, in order to specify a $\mathsf{U}_{\mathsf{PLP}}(\mathcal{B}, \Pi)$. Once one interprets a PLP, another definition is necessary to get the results of the specifications and to eliminate contradictions of the original sets of rules.

In general, the interpretations of an update program $\mathsf{U}_{\mathsf{PLP}}$ come from the answer sets of its corresponding PLP. Such an interpretation shall lead to one or more possible new knowledge bases, as expressed in the following definition.

Definition 19 (Possible Resulting Knowledge Base, $\mathsf{S}_{\mathsf{PLP}}$ [Zha06])**.** *Let $\mathsf{U}_{PLP}(\mathcal{B},\Pi)$ be specified as in Definition 18. A set $\mathcal{B}$' of ground literals is called a* possible resulting knowledge base *with respect to $\mathsf{U}_{PLP}(\mathcal{B},\Pi)$, if and only if $\mathcal{B}$' satisfies the following conditions:*

[8] A predicate symbol corresponds to an atom, in my notation.

[9] There is no formal specification when there exist strong-negated atoms. However, supported by the examples, one may state that for every strong-negated atom $\neg\ell$ in the formula, there is a strong-negated $\neg$new-ℓ atom.

1. *if $U_{PLP}(\mathcal{B},\Pi)$ has a consistent answer set $\mathcal{S}$, then $\mathcal{B}' = \{\ell \mid \textsf{new-}\ell \in \mathcal{S}\}$;*
2. *if $U_{PLP}(\mathcal{B},\Pi)$ does not have a consistent answer set (i.e., $U_{PLP}(\mathcal{B},\Pi)$ is not well defined), then $\mathcal{B}' = \mathcal{B}$.*

The name $S_{PLP}(U_{PLP}(\mathcal{B},\Pi))$ denotes the set of all resulting knowledge bases of $U_{PLP}(\mathcal{B},\Pi)$.

Now, let us start with a simple but representative and thorough example (proposed by [Zha06]) that illustrates this process.

Example 12 ([Zha06])**.** *Suppose the initial knowledge base $\mathcal{B} = \{\neg a, b, c\}$ and the update program*

$$\begin{aligned}\Pi = \{\neg b &\leftarrow \text{not } b\\ a &\leftarrow c\}\end{aligned}$$

By Definition 18, the corresponding PLP specification is $U_{PLP}(\mathcal{B},\Pi) = (\Pi^*, \mathcal{N}, <)$, consisting of the following rules:

Initial knowledge: Π_0*:*

$$\neg a \leftarrow \top \qquad b \leftarrow \top \qquad c \leftarrow \top$$

Inertia rules:

$$\begin{aligned}i_1 &: \quad newa \leftarrow a, \text{not } \neg newa\\ i_2 &: \quad \neg newa \leftarrow \neg a, \text{not } newa\\ i_3 &: \quad newb \leftarrow b, \text{not } \neg newb\\ i_4 &: \quad \neg newb \leftarrow \neg b, \text{not } newb\\ i_5 &: \quad newc \leftarrow c, \text{not } \neg newc\\ i_6 &: \quad \neg newc \leftarrow \neg c, \text{not } newc\end{aligned}$$

Update rules:

$$\begin{aligned}u_1 &: \quad \neg newb \leftarrow \text{not } newb\\ u_2 &: \quad newa \leftarrow newc\end{aligned}$$

Rule preferences *By Definition 18, $i_i < u_j$ with $i, j > 0$ are*

$$\begin{array}{lll} N(i_1) < N(u_1) & N(i_1) < N(u_2) & N(i_2) < N(u_1)\\ N(i_2) < N(u_2) & N(i_3) < N(u_1) & N(i_3) < N(u_2)\\ N(i_4) < N(u_1) & N(i_4) < N(u_2) & N(i_5) < N(u_1)\\ N(i_5) < N(u_2) & N(i_6) < N(u_1) & N(i_6) < N(u_2)\end{array}$$

With these specifications, one may compute the two answer sets of Π^**. Namely,*

$$\{\neg a, b, c, newa, newc, newb\} \tag{33}$$
$$\{\neg a, b, c, newa, newc, \neg newb\} \tag{34}$$

However, the prioritised logic program $\mathcal{P}$ *has the unique answer set (33) from its unique reduct because there is only one rule* (u_1) *defeated by* $\Pi_0 \setminus \{u_1\}$ *with which one may establish the relations* $N(i_3) < N(u_1) \in \mathcal{P}$ *and there are no less-preferred rules than* u_i *in* $\mathcal{P}(<)$ *—Definition 15. As a consequence,* $\mathsf{S}_{PLP}(\mathcal{B}, \Pi) = \{a, b, c\} = \mathcal{S}_{(\Pi_0,\Pi_1)}$ *as expected from Definition 19, and the transformed program from* Π_0 *with respect to* Π_1 *that is a maximal subset of* Π_0 *and is coherent with* $\mathcal{S}_{(\Pi_0,\Pi_1)}$ *is just* $\{b, c\}$*. From this program, the* $\mathsf{U}_Z(\Pi_0, \Pi_1)$ *specification corresponds to the following* $\mathcal{P}$*:*

$$\begin{array}{lll} & \rho_1: & \neg b \leftarrow \text{not } b \\ & \rho_2: & a \leftarrow c \\ & \rho_3: & b \leftarrow \top \\ & \rho_4: & c \leftarrow \top \\ \rho_1 < \rho_3 & \rho_1 < \rho_4 & \\ \rho_2 < \rho_3 & \rho_2 < \rho_4 & \end{array}$$

whose unique answer set out of the unique reduct ***is*** $\{a, b, c\}$*. That is because defeated rules may not derive from a simple fact update, and here is the difference with the extended simple fact update.*

This simple-fact update approach originally appeared in [MT94, MT98]. However, it is not adequate for practical applications for the simple reason that, as its name suggests, its definition does not deal with general (non-factual) rules. As a result, the author in [Zha06] reformulates the approach to allow updating ELP's, rather than only facts, by means of the following two definitions, where the first one eliminates contradictory rules between Π_0 and Π_1.

Definition 20 (Transformed Program, $\Pi_{(\Pi_0,\Pi_1)}$ [Zha06])**.** *Given two consistent programs* Π_0 *and* Π_1*, with* $\mathcal{S}_{\Pi_0}$ *as an answer set of* Π_0 *and* $\mathcal{S}_{(\Pi_0,\Pi_1)}$ *as an answer set of the update of* $\mathcal{S}_{\Pi_0}$ *with* Π_1*. Suppose* $\mathcal{S}_{(\Pi_0,\Pi_1)} \in \mathsf{S}_{PLP}(\mathsf{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1))$*. An extended logic program* $\Pi_{(\Pi_0,\Pi_1)}$ *is called a* transformed program *from* Π_0 *with respect to* Π_1*, if* $\Pi_{(\Pi_0,\Pi_1)}$ *is a maximal subset of the ground instantiation of* Π_0 *such that* $\mathcal{S}_{(\Pi_0,\Pi_1)}$[10] *is coherent with* $\Pi_{(\Pi_0,\Pi_1)}$*.*

Once there is a transformed program, a set of preferences between its rules is to solve possible conflicts.

[10] Note that the original definition must have a typographical error when reading $\mathcal{S}_{\Pi_0}$ rather than $\mathcal{S}_{(\Pi_0,\Pi_1)}$.

Definition 21 (Update Specification, $\mathsf{U}_Z(\Pi_0, \Pi_1)$ [Zha06])**.** *Let $\Pi_{(\Pi_0,\Pi_1)}$ be defined as in Definition 20. A* specification of updating Π_0 with Π_1 *is a PLP, denoted as $\mathsf{U}_Z(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$, where, for each rule ρ in Π_1 and each rule ρ' in $\Pi_{(\Pi_0,\Pi_1)}$, there is a preference relation $\mathcal{N}(\rho) < \mathcal{N}(\rho')$.*

Up to now, a transformed program can eliminate contradictions between an original knowledge base and its update. On the other hand, there are circumstances that do not cause contradiction, but *indefinite results* that must be observed.

5.4 Solving Conflicts

In the process of updating a knowledge base with a logic program, there are rules that might cause conflict when producing indefinite results. The way in which [Zha06] deals with this problem is by overriding old conflicting rules with the new ones, coded in the preferences of a transformed program, and by producing a called *possible resulting program*.

Definition 22 (Possible Resulting Program [Zha06])**.** *A program Π_0' is a* possible resulting program of $\mathsf{U}_Z(\Pi_0, \Pi_1)$ *after updating Π_0 with Π_1 if Π_0' is a reduct of the ground instantiation of $\mathsf{U}_Z(\Pi_0, \Pi_1)$.*

A mandatory test is Example 3, which produces *counterintuitive results* in many of the existing semantics for updates. So, I will compute it under [Zha06]'s approach as follows.

Example 13. *Suppose an initial program*

$$\begin{aligned} \Pi_0 = \{ day &\leftarrow \text{not } night \\ night &\leftarrow \text{not } day \\ stars &\leftarrow night, \text{ not } cloudy \\ \neg stars &\leftarrow \top \} \end{aligned}$$

updated with

$$\begin{aligned} \Pi_1 = \{ stars &\leftarrow constellations \\ constellations &\leftarrow stars \} \end{aligned}$$

Its corresponding PLP specification, $\mathsf{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1) = (\Pi^*, \mathcal{N}, <)$, is as follows:

Initial Knowledge:

$$\begin{aligned} i_0 : &\quad day \leftarrow \top \\ i_0 : &\quad \neg stars \leftarrow \top \end{aligned}$$

Inertial Rules:

$$
\begin{aligned}
i_1 &: \quad newday \leftarrow day, \text{not } \neg newday \\
i_2 &: \quad \neg newday \leftarrow \neg day, \text{not } newday \\
i_3 &: \quad newstars \leftarrow stars, \text{not } \neg newstars \\
i_4 &: \quad \neg newstars \leftarrow \neg stars, \text{not } newstars \\
i_7 &: \quad newconstellations \leftarrow constellations, \text{not } \neg newconstellations \\
i_8 &: \quad \neg newconstellations \leftarrow \neg constellations, \text{not } newconstellations
\end{aligned}
$$

Update Rules:

$$
\begin{aligned}
u_1 &: newstars \leftarrow newconstellations \\
u_2 &: newconstellations \leftarrow newstars
\end{aligned}
$$

Rule Preferences:

$$
\begin{array}{ll}
N(i_1) < N(u_1) & N(i_1) < N(u_2) \\
N(i_2) < N(u_1) & N(i_2) < N(u_2) \\
\vdots & \\
N(i_8) < N(u_1) & N(i_8) < N(u_2)
\end{array}
$$

where its unique $\mathsf{S}_{PLP}(\mathsf{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1)) = \{day, \neg stars\}$. *In this case,* $\Pi_{(\Pi_0,\Pi_1)}$ *coincides with* Π_0 *because* $\mathsf{S}_{PLP}(\mathsf{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1))$ *is coherent with* Π_0*—resp.* $\Pi_{(\Pi_0,\Pi_1)}$*, where*

$$
\begin{aligned}
e(\Pi_0, \mathsf{S}_{PLP}(\mathsf{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1))) = \{ & day \leftarrow \text{not } night \\
& stars \leftarrow night, \text{ not } cloudy \\
& \neg stars \leftarrow \top \}
\end{aligned}
$$

and its answer set is $\{day, \neg stars\}$*, which is consistent with* $\mathsf{S}_{PLP}(\mathsf{U}_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1))$*. Thus,* $\Pi_{(\Pi_0,\Pi_1)}$ *is a maximal subset of* Π_0*.*

Finally, its update specification $\mathsf{U}_Z(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$*, whose possible resulting program is just*

$$
\begin{aligned}
\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)} \setminus \{night \leftarrow \text{not } day\} = & \\
\{ stars & \leftarrow constellations \\
constellations & \leftarrow stars \\
day & \leftarrow \text{not } night \\
stars & \leftarrow night, \text{not } cloudy \\
\neg stars & \leftarrow \top \}
\end{aligned}
$$

with its expected ***answer set*** $\{day, \neg stars\}$.

Despite this nice behaviour, one of the counter-intuitive examples to [Zha06]'s approach has to do with solving conflicts between rules and not with models, where most of the current semantics differ:

Example 14. *Suppose an initial knowledge base* $\Pi_0 = \{p \leftarrow \text{not } q\}$ *being updated with* $\Pi_1 = \{q \leftarrow \text{not } p\}$. *The update specification corresponds to* $U_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1) = (\Pi^*, \mathcal{N}, <)$*, where*

Initial Knowledge:

$$i_0: \quad p \leftarrow \top$$

Inertial Rules:

$$\begin{aligned}
i_1: &\quad newp \leftarrow p, \text{not } \neg newp \\
i_2: &\quad \neg newp \leftarrow \neg p, \text{not } newp \\
i_3: &\quad newq \leftarrow q, \text{not } \neg newq \\
i_4: &\quad \neg newq \leftarrow \neg q, \text{not } newq \\
i_5: &\quad newp \leftarrow p, \text{not } \neg newp \\
i_6: &\quad \neg newp \leftarrow \neg p, \text{not } newp
\end{aligned}$$

Update Rule

$$u_1: \quad newq \leftarrow \text{not } newp$$

Preferences

$$\begin{aligned}
N(i_1) < N(u_1) \qquad & N(i_2) < N(u_1) \\
N(i_3) < N(u_1) \qquad & N(i_4) < N(u_1) \\
N(i_5) < N(u_1) \qquad & N(i_6) < N(u_1)
\end{aligned}$$

So, $S_{PLP}(U_{PLP}(\mathcal{S}_{\Pi_0}, \Pi_1)) = \{p\}$ *and* $\Pi_{(\Pi_0,\Pi_1)} = \Pi_0$*, where* $U_Z(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0,\Pi_1)}, \mathcal{N}, <)$ *whose reduct* $q \leftarrow \text{not } p$ *is the most preferred one and does not coincide with our intuition.*

Last, besides not satisfying some of the postulates already pointed out, the major drawback of this approach is being limited to only one update to a knowledge base. Namely, it is undefined neither for update sequences nor for *successive updates*, which does not seem to lead to immediate practical applications.

6 Conclusions

This report has presented a survey of semantics in ASP that have made long way by going from *simple-fact updates* of logic programs, to updates of unlimited programs in a sequence. Some of these works present a vast collection of postulates and principles, and/or implementation. However, all the proposals here introduced still present drawbacks either for limiting to *one-step update*, or for relying on *syntactical changes* to the original logic program that leads to *counterintuitive results*, which suggests challenging areas of research. .

7 Acknowledgement

I am very grateful to Wojciech Jamroga and Jürgen Dix for their discussions, many useful comments and for proof-reading this report to ensure its optimal quality.

References

[ABBL05] José Júlio Alferes, Federico Banti, Antonio Brogi, and João Alexandre Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1):7–32, 2005.

[AGM85] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2):510–530, June 1985.

[ALP+98] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Teodor C. Przymusinski, and Przymusinska Halina. Dynamic logic programming. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 98–111, San Francisco, June 2–5 1998. Morgan Kaufmann Publishers.

[ALP+99] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, 45(1–3):43–70, 1999.

[APPP02] José Júlio Alferes, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. LUPS —A language for updating logic programs. *Artificial Intelligence*, 138(1–2):87–116, June 2002.

[CZ05] Vino Fernando Crescini and Yan Zhang. Policy updater: A system for dynamic access control. *International Journal of Information Security*, 5(3):145–165, 2005.

[EFLP02] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Computing preferred answer sets by meta-interpretation in answer set programming. Technical Report INFSYS RR-1843-02-01, Vienna University of Technology, 2002.

[EFST00a] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On updates of logic programs: Semantics and properties. Technical Report INFSYS RR-1843-00-08, TU Wien, Institute für Informationssysteme, 2000.

[EFST00b] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. Considerations on updates of logic programs. In Manuel Ojeda-Aciego, Inman P. de Guzmán, Gerhard Brewka, and L. Moniz Pereira, editors, *Logics in Artificial Intelligence, European Workshop, JELIA 2000*, pages 2–20, Malaga, Spain, 2000. Springer Verlag.

[EFST01] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. A framework for declarative update specifications in logic programs. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI*, volume I, pages 649–654, Seattle, Washington, 2001. Morgan Kaufmann.

[EFST02] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming*, 2(6):711–767, 2002.

[EFST05] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. Reasoning about evolving nonmonotonic knowledge bases. *ACM Transactions on Computational Logic*, 6(2):389–440, 2005.

[Fag95] Ronald Fagin. *Reasoning about Knowledge*. The MIT Press, Cambridge, Massachusetts, USA, 1995.

[FKUV86] Ronald Fagin, Gabriel M. Kuper, Jeffrey D. Ullman, and Moshe Y. Vardi. Updating logical databases. *Advances in Computing Research*, 3:1–18, 1986.

[FUV83] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi. On the semantics of updates in databases. In *PODS '83: Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 352–365, New York, NY, USA, 1983. ACM Press.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium ICLP/SLP*, Seattle, Washington, 1988. MIT Press.

[IS95] Katsumi Inoue and Chiaki Sakama. Abductive framework for nonmonotonic theory change. In *the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 204–210, Montreal, Canada, 1995. Morgan Kaufmann Publishers.

[KKT98] A. Kakas, R. Kowalski, and F. Toni. The role of abduction in logic programming. In Dov M. Gabbay, Christopher John Hogger, and John Alan Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, Logic programming, pages 235–324. Oxford University Press, Oxford, UK, 1998.

[KLM90] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, 44(1):167–207, 1990.

[KM89] Hirofumi Katsuno and Alberto O. Mendelzon. A unified view of propositional knowledge base updates. In Sridharan N S, editor, *the 11th International Joint Conference on Artificial Intelligence, IJCAI-89*, pages 1413–1419, Detroit, Michigan, USA, 1989. Morgan Kaufmann.

[KM90] Antonis C. Kakas and Paolo Mancarella. Generalized Stable Models: A semantics for abduction. In *ECAI*, pages 385–391, Stockholm, Sweden, 1990. Aiello L.

[KM91] Hirofumi Katsuno and Alberto O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52(3):263–294, 1991.

[Leh92] Daniel Lehmann. Plausibility Logic. In E. Börger, G. Jäger, H. Kleine-Büning, and M. M. Richter, editors, *Computer Science Logic, 5th Workshop, CSL 91, Berne, Switzerland*, LNCS 626, pages 227–241, Berlin, 1992. Springer.

[LM92] Daniel Lehmann and Menachem Magidor. What does a conditional knowledge base entail? *Artificial Intelligence*, 55:1–60, 1992.

[Mak88] David Makinson. General theory of cumulative inference. In *Non-Monotonic Reasoning, 2nd International Workshop*, pages 1–18, Grassau, Federal Republic of Germany, 1988. Springer.

[Mak94] David Makinson. General Patterns in Nonmonotonic Reasoning. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Nonmonotonic and Uncertain Reasoning*, volume 3, chapter 3, pages 35–110. Oxford University Press, Oxford, UK, 1994.

[MT94] Victor Wiktor Marek and Miroslaw Truszczynski. Revision specifications by means of programs. In *Logics in Artificial Intelligence, European Workshop, JELIA*, pages 122–136, York, UK, September 1994. Springer.

[MT98] Victor Wiktor Marek and Miroslaw Truszczynski. Revision programming. *Theoretical Computer Science*, 190(2):241–277, January 1998.

[OZ03] Mauricio Osorio and Fernando Zacarías. Irrelevance of syntax in updating answer set programs. In *Proceedings of the Fourth Mexican International Conference on Computer Science (ENC' 03) In Workshop on Logic and Agents*, Apizaco, Mexico, 2003. IEEE Computer Society.

[Poo88] David Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

[SI99] Chiaki Sakama and Katsumi Inoue. Updating extended logic programs through abduction. In Michael Gelfond, Nicole Leone, and Gerald Pfeifer, editors, *LPNMR '99: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1730 of *LNCS*, pages 147–161, El Paso, Texas, USA, 1999. Springer-Verlag.

[SI03] Chiaki Sakama and Katsumi Inoue. An abductive framework for computing knowledge base updates. *Theory Pract. Log. Program.*, 3(6):671–715, 2003.

[Win90] Marianne Winslett. *Updating logical databases*. Cambridge University Press, New York, NY, USA, 1990.

[ZF95] Yan Zhang and Norman Foo. On propositional knowledge base updates. *Australian Journal of Intelligent Information Processing Systems*, 2:20–29, 1995.

[ZF97] Yan Zhang and Norman Y. Foo. Answer sets for prioritized logic programs. In *ILPS '97: Proceedings of the 1997 international symposium on Logic programming*, pages 69–83, Cambridge, MA, USA, 1997. MIT Press.

[ZF05] Yan Zhang and Norman Foo. A unified framework for representing logic program updates. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 707–713, Pittsburgh, Pennsylvania, USA, 2005. AAAI Press / The MIT Press.

[Zha01] Yan Zhang. The complexity of logic program updates. In *AI '01: Proceedings of the 14th Australian Joint Conference on Artificial Intelligence*, pages 631–642, London, UK, 2001. Springer-Verlag.

[Zha03] Yan Zhang. Two results for prioritized logic programming. *Theory and Practice of Logic Programming*, 3(2):223–242, March 2003.

[Zha06] Yan Zhang. Logic program-based updates. *ACM Transactions on Computational Logic*, 7(3):421–472, July 2006.